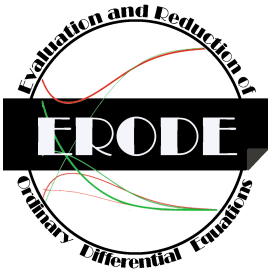
ERODE: Evaluation and Reduction of Ordinary Differential Equations

https://sysma.imtlucca.it/tools/erode/



v 1.0 — June 20, 2018

1 Introduction

ERODE is a software tool for the evaluation and reduction of systems of explicit first-order, autonomous ordinary differential equations (ODEs) that implements the minimization algorithms published in [1, 2, 3, 4]. *ERODE* has been presented in [5], while tutorials on it have been given in [6, 7]. At the basis of *ERODE* are *forward* and *backward differential equivalence*, two complementary equivalence relations over the variables of an ODE system. Forward differential equivalence (FDE) identifies a partition of the ODE variables for which a self-consistent aggregate ODE system can be provided which preserves the sums of variables within a block. *Backward differential equivalence* (BDE) identifies a partition whereby variables in the same block have the same solution whenever initialized equally. The minimization algorithms are partition-refinement algorithms that compute the largest equivalence that refines a given initial partition of variables.

ERODE accepts input ODE systems in two formats: a *direct* specification of an ODE system (close to its mathematical definition) as a collection of functions, each giving the derivative for each variable; and a *reaction-network* (RN) specification, akin to a formal chemical reaction network. For instance, the simple specification $A + B \xrightarrow{k} C$ describes the dynamics of species/variables A, B, C according to *mass-action* kinetics. The ODEs become $dA/dt = dB/dt = -A \cdot B$, $dC/dt = A \cdot B$.

The choice of the input format may affect which analysis and reduction techniques are available. RNs induce, and can encode, ODE systems with derivatives that are multivariate polynomials of any degree [1]. Such systems can be reduced with the specialized algorithms of [1] (which superseded those presented in [4, 2, 5]) for computing the largest *forward* and *backward RN equivalences*. These are equivalence relations defined *syntactically* on the RN syntax: forward RN equivalence (FE) charactirizes FDE for this class of ODEs. Similarly, backward RN bisimulation (BE) characterizes BDE. Importantly, they enable a significantly

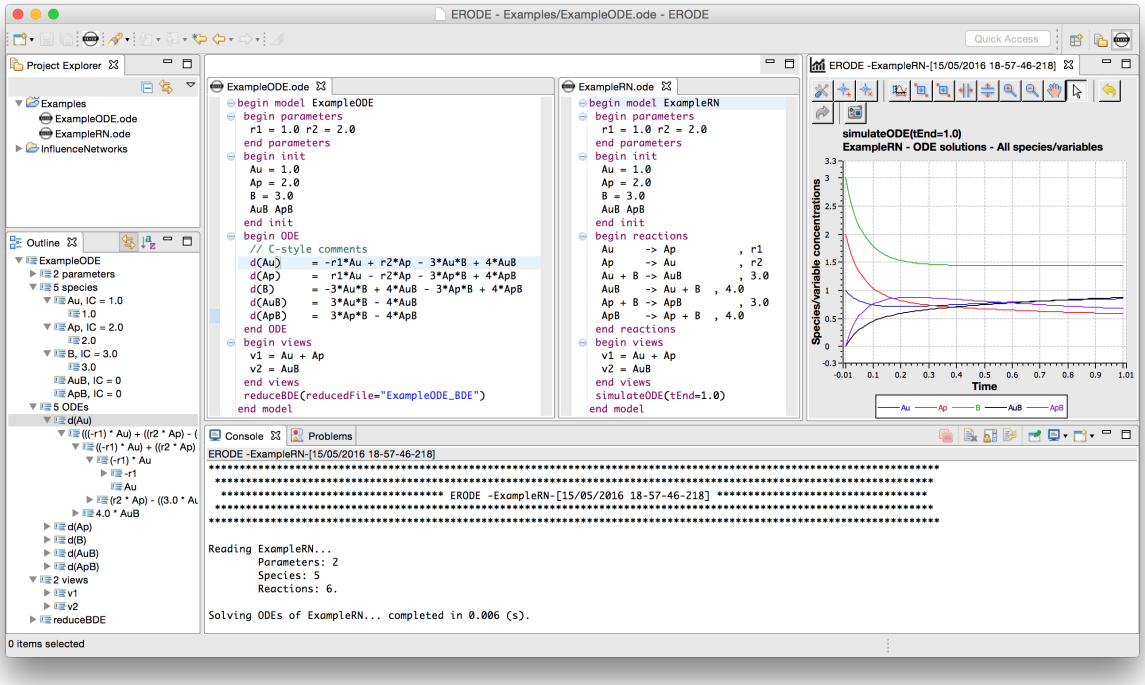


Figure 1: A screenshot of *ERODE*.

faster partition-refinement algorithm than the SMT-based ones for FDE and BDE that are available for a generic input specification, developed in [3].

The outputs of the reduction algorithms are (hopefully) smaller *ERODE* specifications where each macro-variable represents a distinct equivalence class of species. The link between a macro-variable and the members of the corresponding equivalence class is maintained through annotations in the form of comments in the file. The format of the output specification (i.e., plain ODE or RN) is the same as the input.

In addition to ODEs and Reaction Networks ERODE supports the specification of semi-explicit Differential Algebraic Equations (DAEs) in the format presented in [8].

2 Setting up *ERODE*

Installation. *ERODE* is a multi-platform application based on the Eclipse framework. It does not require any installation process. The only requirement is a working installation of **Java 8**, available at:

`https://java.com/en/download/`

ERODE can be downloaded from:

`http://sysma.imtlucca.it/tools/erode/install-erode/`

A screenshot of *ERODE* is shown in Fig. 1.

Updating *ERODE*. *ERODE* is actively developed, and new features will be added. The tool can be updated to the latest version by clicking on **Help** → **Check for Updates**.

Preparing the workspace. Upon installation the user is prompted to select the location for the *workspace*. This is a directory containing all *ERODE* files, arranged into *projects*. *ERODE* recognizes ".ode" files. Projects and *ERODE* files can be created as follows:

1. Create a new *ERODE* project: Right click on the Project Explorer (top-left of Fig. 1), and select **New** → **ERODE Project**. Choose a project name and click **Finish**.

2.A Create an *ERODE* file: Right click on the newly created project in the Project Explorer and choose **New** → **ERODE File**.
Hint: The combo box gives the possibility of generating a basic template file with comments that illustrate various parts of the language.

2.B Alternatively, it is possible to import a file from one of the supported input formats:

`Matlab` : a Matlab function representing the derivatives of an ODE system (extension `.m`);

`BNG` : a CRN generated with the well-established tool BioNetGen version 2.2.5-stable [9] (extension `.net`).

`LBS` : a CRN written in the LBS format of the Microsoft's tool GEC [1] (extension `.lbs`).

Right click on the newly created project in the Project Explorer and choose **New** → **Import BioNetGen File** or **New** → **Import Matlab ODEs**. A dialog will appear allowing first to choose a source folder, and to select a number files contained in there.

3 A dialog will pop-up asking to "add the Xtext nature to the project": Click **Yes**.

Example project. A sample project with *ERODE* examples is available. To use it:

1. Download and decompress the archive available at:

   `http://sysma.imtlucca.it/erode-examples/`

2. Right click on the Project Explorer, and select **Import**. Choose **Existing Projects into Workspace** and click **Next**.

3. Click on **Browse...** and locate the Examples folder. Tick **Copy projects into workspace** and hit **Finish**.

Editing *ERODE* specifications. Upon double-clicking on an *ERODE* file in the Project Explorer, the graphical text editor will open. The key sequence Ctrl+space (Windows/Linux) or Command+space (Mac) will enable content assist, providing contextual code suggestions. In-line markers will be used to highlight errors (and suggest fixes).

Running *ERODE* specification. To run an *ERODE* specification, click the *ERODE* icon in the toolbar ( , top-left of Fig. 1). Alternatively, right-click the file in the Project Explorer, and click **Execute selected ERODE Program.**

[1] http://research.microsoft.com/en-us/projects/gec/

Listing 1: Direct ODE specification.

```
begin model ExampleODE
 begin parameters
  r1 = 1.0 r2 = 2.0
 end parameters
 begin init
  Au = 1.0 Ap = 2.0 B = 3.0
  AuB ApB
 end init
 begin partition
  {Au,Ap}, {AuB}, {B,ApB}
 end partition
 begin ODE
  // C-style comments
  d(Au)   = -r1*Au + r2*Ap - 3*Au*B + 4*AuB
  d(Ap)   =  r1*Au - r2*Ap - 3*Ap*B + 4*ApB
  d(B)    = -3*Au*B + 4*AuB - 3*Ap*B + 4*ApB
  d(AuB)  =  3*Au*B - 4*AuB
  d(ApB)  =  3*Ap*B - 4*ApB
 end ODE
 begin views
  v1 = Au + Ap
  v2 = AuB
 end views
 reduceBDE(reducedFile="ExampleODE_BDE.ode")
end model
```

Listing 2: Reaction network.

```
begin model ExampleRN
 begin parameters
  r1 = 1.0 r2 = 2.0
 end parameters
 begin init
  Au = 1.0 Ap = 2.0 B = 3.0
  AuB ApB
 end init
 begin partition
   {Au,Ap}, {AuB}
  end partition
 begin reactions
  Au     -> Ap      , r1
  Ap     -> Au      , r2
  Au + B -> AuB     , 3.0
  AuB    -> Au + B  , 4.0
  Ap + B -> ApB     , 3.0
  ApB    -> Ap + B  , 4.0
 end reactions
 begin views
  v1 = Au + Ap
  v2 = AuB
 end views
 simulateODE(tEnd=1.0)
end model
```

3 *ERODE* Language

Illustrating example. We show some of *ERODE*'s features using a simple model. This is an idealized biochemical interaction between two molecules, `A` and `B`, where `A` can be in two states (`u` for unphosphorylated and `p` for phosphorylated) undergoing binding/unbinding with `B`. This results in five biochemical *species*: `Au`, `Ap`, `B`, `AuB`, and `ApB`. Each species is associated with one ODE variable which models its concentration as a function of time.

Listings 1 and 2 show the two alternative specification formats for the same model (assuming mass-action kinetics), using plain ODEs or the RN representation, respectively. Listing 3 shows the specification format for a DAE model decsribing a simple RLC circuit.

Specification language. The input format consists of the following parts:

i) Parameter specification;

ii) Declaration of variable names with (optional) initial conditions;

iii) Declaration of (optional) algebraic variable names with (optional) initial conditions;

iv) Initial partition of variables to be given to the reduction algorithms;

v) ODE system, either in plain format or as an RN;

vi) Declaration of (optional) algebraic constraints;

vii) Observables, called *views*, to be tracked by the numerical solvers;

viii) Commands for ODE numerical solution, reduction, and exporting into other formats.

Listing 3: DAE specification.

```
begin model ExampleDAE
 begin parameters
  Vb = 24 L = 1 R = 100 C = 0.001
 end parameters
 begin init
  i_L V
 end init
 begin alginit
  i_R i_C
 end alginit
 begin ODE
  d(V) = i_C/C
  d(i_L) = (Vb - V)/L
 end ODE
 begin algebraic
  i_R = V/R
  i_C = i_L - i_R
 end algebraic
 simulateDAE(tEnd=1.5)
end model
```

Alternatively, points i)-v) can be replaced by a command `importBNG`:

```
importBNG(fileIn=<filename>)
```

used to load "on-the-fly" a CRN generated with BioNetGen 2.2.5.

3.1 Parameter Specification

Parameters are variables that can be used to specify values of initial conditions, interaction rates, or in views. An *ERODE* specification might start with an optional list of numerical parameters enclosed in the `parameters` block. Each is specified as

```
parameterName = expression
```

where `expression` is an arithmetic expression involving parameter names and reals through the following operators: `+`, `-`, `*`, `/`, `^`, `abs`, `min`, and `max`.

3.2 Variable Declaration

The mandatory `init` block defines all ODE variables[2] of the model. Each variable is specified as:

```
<variable> [= IC]
```

where `IC` is an arithmetic expression involving reals and parameter names as above, that evaluates to the initial condition assigned to the variable; `IC` is optional with default value 0. The later use of variables not appearing in this block is considered a syntactic error. As part of its advanced quick-fix framework, *ERODE* collects the list of all such undefined variables, and allows to define them all with one click. This can be done by clicking on the red error mark which appears on the left of the first line of the model specification, as shown in Fig. 2.

[2]Throughout this document we will use the terms 'variable' and 'species' interchangeably.

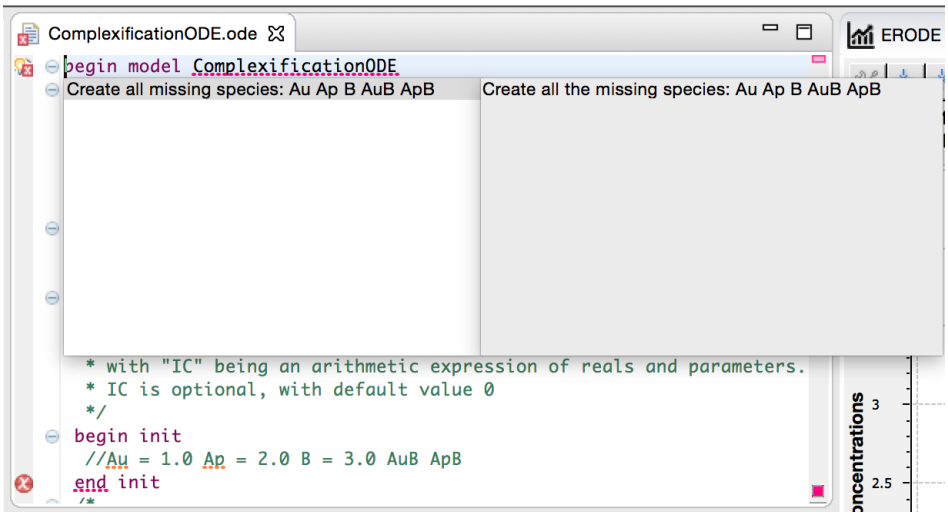


Figure 2: Quick-fix suggestion to automatically declare all undefined variables.

(The tool offers other useful quick-fix suggestions, which can be enabled similarly clicking on the error marks.)

Similarly, when defining a DAE system the `alginit` block defines all the algebraic variables of the model and their optional initial conditions.

3.3 Initial Partition of Variables

As discussed, the minimization techniques implemented by *ERODE* are algorithms that compute the largest equivalence (i.e., the coarsest partition of variables) that refines a given initial partition of variables. For maximal ODE lumping (according to both FDE and BDE) the modeler specifies a singleton partition where all variables are in the same block. However, there might be cases when this is not desirable because:

(FDE) a variable of interest may be aggregated with others; as a result, its individual trace might be lost in the reduction;

(BDE) variables must be pre-partitioned in blocks with same initial concentration.

An initial partition of variables can be specified in the optional `partition` block. This can then be used in the reduction commands, as described later. The user is required to specify only the partition blocks of interest, while all not mentioned variables are assigned to an implicit additional block. For instance, Listings 1 and 2 represent the same initial partition {{`Au,Ap`},{`AuB`},{`B,ApB`}}.

3.4 ODE Definition

Direct ODE declaration. In the direct declaration format the derivatives are specified within the `ODE` block. Each equation is specified as:

```
d(<variable>) = drift
```

where `drift` is an arithmetic expression containing ODE variables, algebraic variables (in the case of DAE systems), parameters and reals through the operators defined in Section 3.1. This syntax identifies a class of ODEs for which the computation of differential equivalences is decidable [3].

Reaction network format. In the RN format, ODEs are inferred from reactions in the form:

```
reagents -> products, rate [ identifier ]
```

where `reagents` and `products` are two multisets of variables. The multiplicity of a variable in a multiset can be defined through the `+` operator or with the `*` operator in the obvious way; that is, `A + A` is equivalent to `2*A`. In some cases, it might be useful to assign an identifier to some reactions. This can be done by adding `[ identifier ]` after the rate, where `identifier` identifies the reaction.

The semantics of the reaction is that it generates a negative term in the ODE corresponding to each reagent, and a positive term in the ODE of each product, proportionally to their multiplicity. Three kinds of rates are supported:

- **Mass action:** If `rate` is a variable-free expression that evaluates to a real number (as in all reactions of Listing 2), then the reaction represents a dynamics akin to the well-known law of mass action. For example, `Au + B -> AuB, 3.0` leads to a term $+3 \cdot \texttt{Au} \cdot \texttt{B}\cdot$ in the ODE of `AuB`, and to $-3 \cdot \texttt{Au} \cdot \texttt{B}\cdot$ it the ODEs of both `Au` and `B`.

  Note. The parameter `rate` **needs not** evaluate to a positive real: this allows to encode an arbitrary ODE with polynomial derivatives into an RN [4]. When every rate does evaluate to positive real we call the RN a *chemical reaction network* (CRN). It identifies a class of models for which specific analysis options are available in *ERODE*. Furthermore, an *elementary* CRN is one where the size of the reagent multiset is at most two.

- **Arbitrary:** *ERODE* also supports more generic arithmetical expressions for rates through the `arbitrary` keyword. In this case, the reaction firing rate is explicit. For instance,

  ```
  Au + B -> AuB, arbitrary 3.0*Au*B
  ```

  is equivalent to `Au + B -> AuB, 3.0`.

- **Hill:** *ERODE* also allows to specify reactions with Hill kinetics [10]. This is obtained with rates of form `Hill K k R1 r1 R2 r2 n1 a n2 b`, with `k`, `r1` and `r2` being real parameters, and `a` and `b` two naturals. Only binary Hill reactions (i.e., with two reagents) are allowed. The `Hill` keyword is actually treated as syntactic sugar, as Hill reactions are transformed in arbitrary reactions with rate

  $$\frac{\texttt{k} \cdot \texttt{X}^{\texttt{a}} \cdot \texttt{Y}^{\texttt{b}}}{(\texttt{r1} + \texttt{X}^{\texttt{a}}) \cdot (\texttt{r2} + \texttt{Y}^{\texttt{b}})}.$$

3.5 DAE Definition

In the definition of Differential Algebraic Equations systems the declaration of the dynamics of the differential variables are specified in the `ODE` block. The declaration of the algebraic constraints is specified in the `algebraic` block. Each algebraic constraint is specified as:

```
<AlgebraicVariable> = constraint
```

where `constraint` is an arithmetic expression containing differential variables, algebraic variables, parameters and reals through the operators defined in Section 3.1.

3.6 Views

Views represent the variables of interest to the modeller. As for ODEs, each view can be specified as an arithmetic expression involving variables, parameters and reals. In the running example the intent is to collect the total concentration of the `A`-molecules, regardless of their phosphorylation state, and the concentration of the compound `AuB`, respectively.

For a CRN specification, views expressions can also contain terms of form

`var(s1)` and `covar(s1,s2)`

referring, respectively, to the variance of the ODE variable `s1`, and to the covariance of the ODE variables `s1` and `s2`. In other words, *ERODE* implements the so-called *linear noise approximation* of [11], allowing to study not just the first order moments of the ODE variables, but also their variance.

3.7 ODE Solution

The numerical solution of the ODE can be obtained through the `simulateODE` command:

```
simulateODE(tEnd=<value>, steps=<value> , csvFile=<filename>,
            viewPlot=<VARS&VIEWS | VARS | VIEWS | NO >,
            defaultIC= <value>, computeJacobian = <true | false>,
             library = <APACHE | SUNDIALS>)
```

It integrates the ODE system starting from the specified initial conditions up to time point `tEnd`, interpolating the results at `steps` equally spaced time points. If the optional argument `csvFile` is present, the plots are exported into comma-separated values formats. If the optional argument `viewPlot` is not present or it is set to `VARS&VIEWS`, then two plots are generated, one for the the trace of each ODE variable and one for the trace of each specified view, respectively. The optional argument `defaultIC` allows to overwrite the default value of unspecified initial concentrations. The optional argument `library` allows to choose which solver to use. By default, the solver used is provided in the `Apache` library (`http://commons.apache.org/proper/commons-math/`). Otherwise it is possible to select the CVODE solver provided in the library `SUNDIALS` [12].

In the case of elementary RN specifications, the optional argument `computeJacobian` can be used to trigger the evaluation of the Jacobian matrix of the reaction network at each sampled point. The Jacobian will be plotted and/or stored in a csv file depending on the value of the `csvFile` and `viewPlot` arguments.

3.8 DAE Solution

The numerical solution of the DAE systems can be obtained through the `simulateDAE` command:

```
simulateDAE(tEnd=<value>, steps=<value> , csvFile=<filename>,
            viewPlot=<VARS&VIEWS | VARS | VIEWS | NO >,
            defaultIC= <value>)
```

It solves the DAE system starting from the specified initial conditions up to time point `tEnd`, interpolating the results at `steps` equally spaced time points. This is done using the IDA solver provided in the library `SUNDIALS` [12]. If the optional argument `csvFile` is present, the plots are exported into comma-separated values formats. If the optional argument `viewPlot` is not present or it is set to `VARS&VIEWS`, then two plots are generated, one for the the solution of each differential or algebraic variable and one for the trace of each specified view, respectively. The optional argument `defaultIC` allows to overwrite the default value of unspecified initial concentrations.

3.9 CRN Stochastic Simulation

CRNs can also be interpreted as a stochastic system of reactions in terms of a continuous-time Markov chain (CTMC), following an established approach [13]. *ERODE* allows to simulate sample paths, or to analyze the CRN exactly by the underlying forward equations (the *chemical master equation*, CME).

Stochastic simulation. Stochastic simulation is available with the command

```
simulateCTMC(tEnd=<value>,repeats=<value>,method=<simulationMethod>)
```

where `tEnd` represents the time horizon, `repeats` specifies the number of independent simulations to be performed, and `method` allows to choose the simulation algorithm. *ERODE* uses the *FERN* library for stochastic simulation [14], which makes available the following options:

- `ssa`: Gillepsie's direct method;
- `ssa+`: Gillepsie's direct method enhanced using dependency graphs;
- `nextReaction`: Next-reaction method by Gibson and Bruck;
- `tauLeapingAbs`, `tauLeapingRelProp` and `tauLeapingRelPop`: three variants of the Tau-leaping algorithm, providing different error bounds;
- `maximalTimeStep`: Maximal time step method by Puchalka.

Details on all the supported simulation methods can be found in [14].

When more than one sample is requested, *ERODE* computes the average trajectories from time 0 to `tEnd`. The user can specify further options using the arguments `steps`, `viewPlot`, `csvFile` and `defaultIC`, similarly to the ODE numerical solution.

Chemical Master Equation. *ERODE* can explicitly build the CTMC underlying a chemical reaction network through its CME. This is stored in an *ERODE* file as a CRN having one reaction (with unary reagents and unary products) per transition, using the command:

```
generateCME(fileOut=<filename>)
```

If the output *ERODE* file is analyzed with `simulateODE`, this corresponds to numerically integrating the forward equations of the CTMC. As shown in [3, 4] applying FDE/FE and BDE/BE corresponds to minimizing the CTMC according to the well-known notions of ordinary and exact lumpability for CTMCs [15], respectively.

3.10 Reduction Commands

All ODE reduction commands share the common signature

```
reduce<kind>(prePartition=<NO | IC | USER>, reducedFile=<name>)
```

where `kind` can be `FDE`, `BDE`, `FE`, or `BE`. The ODE input format affects which reduction options are available. For an ODE system defined directly, only FDE and BDE are enabled. FE and BE are additionally available for reaction networks yielding polynomial ODE systems of any degree [1]. This is imposed by restricting to mass-action type rate expressions.

In addition, ERODE supports a further reduction technique for the stochastic semantics of (chemical) reaction networks [13]:

- `reduceSMB`: an RN equivalence in the style of FE which induces ordinary lumpability on the CME underlying an elementary mass-action CRN [16].

The option `prePartition` defines the initial partition passed to the minimization algorithm. The maximal aggregation is obtained with the `NO` option. If it is set to `IC`, the initial partition to be refined is built according to the constraints given by the initial conditions: variables are in the same initial block whenever their initial conditions (specified in the `init` block) are equal. If the option is set to `USER`, then the partition specified in the `partition` block will be used. In the case of BDE/BE the reduction may not be consistent with the initial condition, in the sense that the user can specify a block with two variables that have different initial conditions, breaking the side condition imposed by these equivalences: *ERODE* will issue a warning to the user in this case.

The argument `reducedFile` generates a reduced model in the same format as the input, following the model-to-model transformation algorithms presented in [1] (for FE and BE) and [3] (for FDE and BDE). In all cases, in the reduced *ERODE* model there will be one variable for each computed equivalence class. The name of the variable is (arbitrarily) given by the first variable name in that block, according to a lexicographical order. All members of each equivalence class are reported as annotations in comments. For the BDE case, each reduced variable represents any single variable of the corresponding equivalence class. Instead, in the other cases (FE, BE, and FDE) each variable represents the sum of all variables in the corresponding equivalence class.

3.11 Exporting Options

Conversion options. An explicit ODE specification can be converted in the RN format (and vice versa) using

```
write(fileOut=fileName,format=<ODE | NET |MA-RN>)
```

If the `format` option is set to `ODE`, then the target file will be in explicit ODE format, while with the `RN` option an RN will be generated. If the specification to be exported is an explicit ODE with derivatives given by multivariate polynomials of degree at most two, the `MA-RN` will use the encoding of [4] to output a mass-action RN.

Export to third-party languages. The command:

```
export<format>(fileIn=fileName)
```

exports *ERODE* files into four different target third-party languages:

`Matlab` : a Matlab function representing the derivatives of an ODE system (extension `.m`);

`BNG` : a CRN generated with the well-established tool BioNetGen version 2.2.5-stable [9] (extension `.net`). Available for CRN specifications only;

`SBML` : the well-known `SBML` interchange format (http://sbml.org) (extension `.sbml`).

`LBS` : Exporting support is offered for the LBS format of the Microsoft's tool GEC [3] (extension `.lbs`). Available for CRN specifications only. *ERODE*'s analysis options are not translated into LBS directives.

`Modelica` : a Modelica model representing the ODE/DAE system compatible with OpenModelica (extension `.mo`).

3.12 Further Commands

It is possible to change the value of a parameter or the initial concentration of a variable:

```
setIC(species=x0,expr=1)        setParam(param=p1,expr=2)
```

These are useful in case one is interested in considering different variants of a given *ERODE* specification.

4 Examples

In this section we show a number of scenarios using variants of the running example.

[3] http://research.microsoft.com/en-us/projects/gec/

Listing 4: FDE reduction.

```
begin model ExampleODE_FDE
 begin parameters
  r1 = 1.0
  r2 = 2.0
 end parameters
 begin init
  Au  = 1.0 + 2.0
  B   = 3.0
  AuB
 end init
 begin ODE
  d(Au)   = - 3*Au*B + 4*AuB
  d(B)    = - 3*Au*B + 4*AuB
  d(AuB)  =   3*Au*B - 4*AuB
 end ODE
 //Comments associated to the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

Listing 5: BDE reduction.

```
begin model ExampleODE_BDE
 begin parameters
  r1 = 1.0
  r2 = 1.0
 end parameters
 begin init
  Au  = 1.0
  B   = 3.0
  AuB
 end init
 begin ODE
  d(Au)   = - 3*Au*B + 4*AuB
  d(B)    = - 6*Au*B + 8*AuB
  d(AuB)  =   3*Au*B - 4*AuB
 end ODE
 //Comments associated to the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

Direct specification/FDE. The running example can be reduced according to FDE. Indeed, it is not difficult to see that one can write an ODE system for `B` and the sum of variables `Au+Ap` and `AuB+ApB`, corresponding to the FDE partition {{`Au,Ap`},{`B`},{`AuB,ApB`}}.

However, this reduction is not found by *ERODE* if the pre-partitioning flag is set to `USER`. In fact, the above FDE partition is not a refinement of the user specified one, where `AuB` is in a singleton block. The output file for the case without pre-partitioning is shown in Listing 4, which also shows how the association between the original ODE variables and those in the reduced model is maintained by automatically annotating the output file with comments alongside the new variables.[4]

Direct specification/BDE. The partition {{`Au,Ap`},{`B`},{`AuB,ApB`}} can also be shown to be a BDE if `r1` is equal to `r2`. Intuitively, this would capture the fact that the phosphorylation state of `A` does not impact the way in which the molecules react within this model. Thus, it would be enough to consider either state as a representative of the dynamics. However, if BDE is run with pre-partitioning set ot `IC`, this partition violates the initial conditions for `Au` and `Ap`; indeed, no reduction is found in this case. Instead, if the pre-partitioning is disabled, then the above partition is the coarsest refinement; the user is warned about the inconsistency with the initial conditions. The BDE reduction algorithm rewrites the ODEs of the representatives of each equivalence class. The initial condition for the ODE of the representative is equal to that of the corresponding variable in the original model. The BDE reduction without pre-partitioning for `r1=1.0` and `r2=1.0` is given in Listing 5.

RN/Equivalences. When the input is an elementary RN, the reduction algorithm for FE and BE of [1] can be used. It is based on Paige and Tarjan's coarsest-refinement algorithm [17] and quantitative extensions thereof [18]. The FE and BE equivalences of the model of Listing 2 are the same as those computed using FDE and BDE. For example, the result of BE reduction is provided in Listing 6. As for BDE, we considered the case $r1 = 1.0$ and $r2 = 1.0$

[4]Output files have been typographically adjusted to improve presentation.

Listing 6: BE reduction.

```
begin model ExampleRN_BE
 begin parameters
  r1 = 1.0   r2 = 1.0
 end parameters
 begin init
  Au = 1.0+2.0   B = 3.0   AuB
 end init
 begin reactions
  Au + B -> AuB , 3.0
  AuB -> Au + B , 4.0
 end reactions
 //Comments associated to the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

without pre-partitioning. The comments at the end of the model confirm that we computed the same equivalence as in Listing 5. However, it can be shown that the underlying ODEs of the reduced model do not correspond to those of Listing 5. This is expected, because as discussed in Section 3.10, in a BDE reduced model each variable describes any single variable of the corresponding equivalence class. Instead, in a BE reduced model each variable describes the sum of all variables in the corresponding equivalence class.

References

[1] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Maximal aggregation of polynomial dynamical systems. *National Academy of Sciences. Proceedings*, 2017.

[2] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Forward and backward bisimulations for chemical reaction networks. In *CONCUR*, 2015.

[3] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Symbolic computation of differential equivalences. In *POPL*, 2016.

[4] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Efficient syntax-driven lumping of differential equations. In *TACAS*, 2016.

[5] Luca Cardelli, Mirco Tribastone, Andrea Vandin, and Max Tschaikowski. ERODE: A tool for the evaluation and reduction of ordinary differential equations. In *Tools and Algorithms for the Construction and Analysis of Systems — 23rd International Conference, TACAS*, 2017.

[6] Andrea Vandin and Mirco Tribastone. Quantitative abstractions for collective adaptive systems. In *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM*, pages 202–232, 2016.

[7] Mirco Tribastone and Andrea Vandin. Speeding up stochastic and deterministic simulation by aggregation: an advanced tutorial. In *2018 Winter Simulation Conference (WSC)*, 2018.

[8] G. Yu. Kulikov. Numerical methods solving the semi-explicit differential-algebraic equations by implicit multistep fixed stepsize methods. *Korean Journal of Computational & Applied Mathematics*, 4(2):281–318, 1997.

[9] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

[10] Eberhard O. Voit. Biochemical systems theory: A review. *ISRN Biomathematics*, 2013:53, 2013.

[11] Luca Bortolussi and Roberta Lanciani. *Quantitative Evaluation of Systems: 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, chapter Model Checking Markov Population Models by Central Limit Approximation, pages 123–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[12] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[13] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[14] Florian Erhard, Caroline C. Friedel, and Ralf Zimmer. FERN - a Java framework for stochastic simulation and evaluation of reaction networks. *BMC Bioinformatics*, 9(1):356, 2008.

[15] Peter Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, 31(1):59–75, 1994.

[16] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Syntactic markovian bisimulation for chemical reaction networks. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 466–483, 2017.

[17] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[18] Antti Valmari and Giuliana Franceschinis. Simple $o(m \log n)$ time Markov chain lumping. In *TACAS*, 2010.